Setup

Proxy settings

1. Using the applications own proxy settings (if available) A good first step is to check if the application itself supports proxy. If it does, and it communicates over HTTPS you can proxy the traffic to burp. Remeber that if the application is running over HTTPS you need to install your burp root certificate if you want to be able to intercept the traffic.
2. If the application does not provide their own proxy settings you can set them in the OS. On windows you do this by searching for "Proxy settings" in the search bar. Then you click on "Lan manager" or something like that. There you can define your port and ip.
3. Using TCP-dump If it is not possible to to set the OS proxy you can use tcpdump, and dump the traffic to a pcap file, and then load it up in burp. In order to load it up in burp you need need the Burp extender "Pcap importer", remember that you can only use this for HTTP traffic, and not HTTPS traffic.

Decompile the executable

If the application is built in .Net you can use the program ILSpy to decompile it, to read the source-code.

Things to test

http/https SQL-authentication? SQL injection? Authentication

Recon

- Is the application three-tier (client-server-database) or two tier (client-database)?
- Language used
- Check if the application is proxy-aware or not. Note that an application might be proxy aware for most parts, but not all parts. So run all traffic through an invisible proxy.

Follow these instructions: https://docs.mitmproxy.org/stable/howto-transparent-vms/

-

Network communication

Sniff traffic while using the application Use Wireshark and Sysinternals TCPView.

- Identify protocols used by the application - is there anything more than just http?
- Check if anything is transfer unencrypted
- Check that the applications updating method is run over HTTPS
- Check that the updating process is actually validating the server certificate.

Client side

Check for sensitive information

- Check for sensitive information contained in the application. It might be API keys, username/password, connection strings.
  - USe procmon to see which files are being used by the application.
- Check the configuration files for sensitive data
- Check the registry for sensitive data
  - Use regshot maybe.
- Check the source code for sensitive information. It might be hardcoded passwords or things like that.
- Check for database credentials in memory. It is common tha two-tier applications store database credentials in memory. (Check memory) [https://resources.infosecinstitute.com/damn-vulnerable-thick-client-app-part-3/#article]

Data storage

- Is data cached or stored on the computer? Making it available to unauthenticated users with access to the computer?

Local privileged escalation

- Check for DLL hijacking privilege escalation

Access control

- Check that access control is performed on the backend side and not only on the front-end.

Injection

- CSV - is it possible to import a CSV file?

Server side attacks

All the standard OWASP stuff. XML, injections, access control etc.

Misc

- If all actions in the system are logged, can those logs be manipulated? For example simply by reconfiguring the clock of the computer?
- Check if sensitive information is put in logs on the computer. Information that might be of interest to another user on the computer that has not authenticated to the application.
- Check for hidden functionality - this might be found in the code for example.
- Has the application been compiled with security mechanisms such as DEP and ASLR? Source

References:

https://www.cyberark.com/resources/threat-research-blog/thick-client-penetration-testing-methodology